NAME

pm_dreamer - Optimization of functional forms for particle mechanics.

VERSION

Version 0.2

SYNOPSIS

pm_dreamer input_data beagle_config_file [**-c**] [**-f** *file_header*] [**-g** *start_size end_size*] [**-h**] [**-n** *notice_level*] [**-p** *hfc_copy_num*] [**-r** *rseed*] [**-s** *stat_type*] [**-t** *energy_type*] [**-z**]

DESCRIPTION

PM-Dreamer is software for generation of empirical models for particle mechanics. The software takes as input data from a series of particle configurations and the corresponding energies and/or particle forces associated with those configurations. The output from PM-Dreamer is a set of functions that can potentially be used to calculate configuration energies for particles giving the force-**f**ield necessary for particle simulations.

PM-Dreamer obtains the equations for energy calculation using a combination of genetic programming and local search in order to minimize the root-mean square error in the calculation of energy and/or particle force. The genetic programming is based on the Open-BEAGLE library for evolutionary computation. This library has been extended in PM-Dreamer to allow for massively parellel optimization, hybrid local search, vectorized expression evaluation, template-based evaluation of fitnesses using particle configurations with periodic boundary conditions for 2 and 3-body particle interactions, and parallel restarts with the capability to switch datasets and/or function templates.

The input for PM-Dreamer consists of the *input_data* file that contains particle configuration data and the *beagle_config_file* that facilitates parameterization of the optimization. The formats available for the *input_data* are described in the Fitness Evaluation section and can be specified with the **-t** flag. The format for the *beagle_config_file* is taken from Open Beagle with the extensions described throughout this documentation. Examples for both should have been included with the software package.

There are three types of output for PM-Dreamer. Console output describes the progress and statistics of the run and is controlled with the **-n** flag. Log file output also describes statistics in XML as specified in the *beagle_config_file*. The default filename for the log file for serial runs is gp_force.log. For parallel runs a separate log file is written for each process with the default name gp_force_RR.log where RR is the process rank. The final output format consists of milestone files. These files contain an XML description of all of the expressions at a given point in the optimization and are also used to restart runs. The default name for the milestone files is gp_force.obm for serial or gp_force_RR.obm for parallel runs. Utilities for generating plots from the log files and graphic representations of expressions should have been included with this software package.

The following definitions are used throughout the documentation:

Individual A single mathematical expression for calculation of the energies of particle configurations.

Primitive A primitive is a node in the expression tree. Examples of primitives include unary and binary mathematical expressions, variables used to describe the particle configurations, and constants in the expression.

Terminal/Constant/Ephemeral A terminal is a primitive that takes no arguments. A constant/ephemeral is a terminal that is not used as a variable in the expression. Constants are typically randomly generated and can change during the optimization by mutation to generate a new random number or by local search executed to optimize the constants in an expression. Constants that should not be modified (such as pi) can also be specified.

Fitness A metric describing the error in an Individuals' calculation of the energy using the training data.

Population/Deme A population or deme is a group of individuals that evolve together. Crossover occurs using multiple individuals from a population and selection occurs based on the individuals in a single population. Multiple populations can be used in a run. The populations evolve separately, but can interact through migration of individuals between the populations.

Island Here, an island is used in parallel runs to describe the population or set of populations undergoing evolution in a single MPI process.

Vivarium All of the populations involed in a run.

Hall of Fame The Hall of Fame contains the *n* individuals with the best fitness(es) found during a run.

Milestone/Restart File These files contain the output of all of the individuals at some point in the optimization and have the extension .obm

Hybrid Optimization/Local Search Hybrid optimization occurs separately from the evolutionary optimization. With a specified probability, local search is performed on an individual to optimize 1 or multiple constants.

PARAMETERS

-c Restart from existing milestone files. When restarting, the functional form (**-t**), the *beagle_config_file* file, and the *input_data* can be different from those used in the original run. This allows the user to change parameters and/or add data to refine runs. If the **-f** flag was used to specify a non-default file header for the restart files. The **-f** flag should also be specified again with the same name when using **-r**. When running in parallel, the same number of processes should be used for the restart. If a smaller number is used, the extra individuals will be ignored. If a larger number is used, an error is generated. When restarting an optimization, the generation number starts at the last generation in the milestone file. Therefore, the maxgens termination criterion may need to be increased. The restart files are read by the ReadRestartOp in the *beagle_config_file*

-f *file_header*

Specify the header for the .log output file and the .obm milestone files. The default is gp_force.

-g *start_size end_size*

Scale the fitness by the number of nodes in the tree. This can be used to reduce the average size of individuals. A tree with *start_size* or smaller nodes has a maximum fitness of 1.0. A tree with *end_size* or greater nodes has a maximum fitness of 0.0.

-h Print out the man page for help

-n *notice_level*

Set the degree of program output. Use:

-n 0 No output
-n 10 Normal program output
-n 20 Parameters useful for reproducing the results
-n 30 All output. The degree of Open Beagle Output changes at 10,20, and 30.

-p *hfc_copy_num*

Sets the ec.hfc.copy_num register as described below for HFC from the commandline. ec.mig.mpi_split is set to the same value to allow use with the MPI migration operator.

-r *rseed*

Specify the random seed (unsigned long). Default is 1.

-s *stat_type*

Choose the fitness statistic used. Options are RMSD for adaptive RMSD, CORR for the Pearson correlation coefficient, and OLS for ordinary least squares fitting. See the Fitness section for details on each method.

-t *energy_type*

Specify the functional form of the energy function

-z Disable vectorized tree evaluation. This will typically be at least 4x slower for the optimized Beagle library and greater than 15x slower for the unoptimzed library.

BEAGLE CONFIGURATION FILE

The Beagle configuration file is used to control the optimization including the functions, terminals, operators, and replacement strategies that are used. Deatils on each section follow. A template for a configuration files is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Beagle>
  <Evolver>
    <BootStrapSet>

      ... Population Initialization ...

    </BootStrapSet>
    <MainLoopSet>

      ... Replacement Strategy ...

        ... Fitness Evaluation ...

          ... Crossover ...
```

```
        ... Selection ...

      ... Fitness Evaluation ...

       ... Mutation ...

        ... Selection ...

      ... Selection ...

     ... Migration ...

     ... Statistics ...

     ... Termination Criteria ...

     ... Restart File Output ...

   </MainLoopSet>
  </Evolver>
  <System>
  <PrimitiveSuperSet>
   <PrimitiveSet>

     ... Functions ...

     ... Terminals ...

   </PrimitiveSet>
  </PrimitiveSuperSet>
  <Register>

     ... Register values ...

  </Register>
  </System>
</Beagle>
```

POPULATION INITIALIZATION

The initialization is accomplished using the following operators:

GP-InitHalfOp

Koza's ramped half-and-**h**alf generative method. An equal number of expression trees are generated using a depth parameter that ranges between 2 and the maximum specified depth

GP-InitFullOp

The initial population will consist of expression trees that all have a depth equal to the maximum depth.

GP-InitGrowOp

The initial population consists of expression trees of variable depths.

RestartReadOp

Read in population from a restart (milestone) file. This operator replaces the MilestoneReadOp operator in Open Beagle to allow the parameters in the *beagle_config_file* to override those in the milestone file. The example below checks to see if the register, ms.restart.file, is set. If it is, a restart file is read in. Otherwise, a population is generated using half-and-**h**alf followed by fitness evaluation and statistics output:

```
<BootStrapSet>
  <IfThenElseOp parameter="ms.restart.file" value="">
    <PositiveOpSet>
      <GP-InitHalfOp/>
      <EnergyOp/>
      <GP-StatsCalcFitnessSimpleOp/>
    </PositiveOpSet>
    <NegativeOpSet>
      <RestartReadOp/>
    </NegativeOpSet>
  </IfThenElseOp>
</BootStrapSet>
```

REPLACEMENT STRATEGY AND MIGRATION

The replacement strategy is specified using the follow operators:

DecimateOp

Shrink the population size by keeping the n best individuals

GenerationalOp

Breeding tree following a generation by generation replacement strategy

HierarchicalFairCompetitionOp

HCF operator inspired by the work of Hu and Goodman

MigrationRandomRingOp

Migrate randomly chosen individuals between populations using a ring topology

MuCommaLambdaOp

A (Mu,Lambda) operator generates Lambda children individuals from a population of Mu parents(where Lambda > Mu). From these Lambda individual, it keeps the Mu best to constitute the new generation.

MuPlusLambdaOp

A (Mu+Lambda) operator generates Lambda children individuals from a population of Mu parents (usually where Lambda > Mu). From the Mu parents and the Lambda individual, it keeps the Mu best individuals to constitute the new generation.

NSGA2Op

The NSGA2 replacement strategy implement the elitist multiobjective evolutionary algorithm NSGA2 (Non-dominating Sorting Genetic Algorithm)

OversizeOp

An oversize operator generates (ratio * population size) children individuals from a population of Mu parents.

SteadyStateOp

Steady state replacement strategy operator

CROSSOVER AND MUTATION

Crossover and mutation are specified using the follow operators:

GP-CrossoverOp

Crossover of two individuals to produce a new individual

GP-MutationEphemeralDoubleOp

Mutate the value of a randomly chosen double precision constant in the tree

GP-MutationShrinkOp

Replace a randomly chosen branch with a randomly chosen argument on the branch

GP-MutationStandardOp

Canonical GP Mutation

GP-MutationSwapOp

Swap nodes in the expression tree

GP-MutationSwapSubtreeOp

Swap branches in the expression tree

SELECTION

Selection is specified using the follow operators:

NPGA20p

Multiobjective evolutionary algorithm NPGA 2 (Niched Pareto Genetic Algorithm)

SelectParsimonyTournOp

A simple lexicographic parsimony pressure tournament selection operator, based an idea presented in: Luke, S., and L. Panait. 2002. Lexicographic Parsimony Pressure.

SelectRandomOp

Select an individual in a population randomly operator class (uniform distribution).

SelectRouletteOp

Proportionnal roulette selection operator class.

SelectTournamentOp

Tournament selection operator class.

TERMINATION

Optimization is terminated using the following operators:

TermMaxGenOp

Maximum generation termination criterion operator.

TermMaxFitnessOp

Maximum fitness value termination criterion operator class.

TermMaxHitsOp

Number of hits required in an individual in order for the evolution process to terminate.

TermMaxEvalsOp

Maximum number of fitness evaluations termination criterion operator.

RESTART FILES AND POPULATION OUTPUT

Files output containing populations that can also be used for continuing a simulation are generated with the following operators. (See also POPULATION INITIALIZATION.)

MilestoneWriteOp

Write out a milestone file

ParetoFrontCalculateOp

Evaluate Pareto front from demes and vivarium and put it in place of the actual hall-of-**f**ame. The Pareto front is evaluated just before milestones are written. If previous hall-of-**f**ame are presents in the demes/vivarium, they are erased. This operator must be in the evolver's operator sets between the termination criterion check operators and the MilestoneWriteOp operator.

STATISTICS

Statistics on fitness, function and terminal usage, and expression tree size are generated using the follow operators:

GP-StatsCalcFitnessSimpleOp, GP-StatsCalcFitnessKozaOp, GP-PrimitiveUsageStatsOp, GP-IndividualSizeFrequencyStatsOp

ADF and Constrained Operators

Automatically Defined Functions (ADF) and constrained operators are also available:

GP-ModuleCompressOp, GP-ModuleExpandOp, GP-CrossoverConstrainedOp, GP-InitHalfConstrainedOp, GP-InitFullConstrainedOp, GP-InitGrowConstrainedOp, GP-MutationShrinkConstrainedOp, GP-MutationStandardConstrainedOp, GP-MutationSwapConstrainedOp, GP-MutationSwapSubtreeConstrainedOp.

The additional primitives for the ADF operators include:

ADF (Automatically Defined Function) and ARG (Generic Argument for ADF)

FUNCTIONS

The following functions can be utilized as primitives in the expression

Abs,Add,Cos,Divide,Exp,Log,Multiply,Sin,Subtract

Additional functions added by PM-Dreamer are described below. Functions are added by listing the function name and bias in the primitive set. For example:

```
<Primitive name="ADD" bias="1"/>
```

TERMINALS

The terminals are primitives in the expression tree that do not take arguments (e.g. constants in the expression or variables of the expression. Some that can be included are a double precision number [-1, 1] (E), PI (Pi), and/or a variable, (X), for the potential:

```
<Primitive name="E" bias="1"/>
<Primitive name="Pi" bias="1"/>
<Primitive name="X" bias="1"/>
```

ADDITIONAL PRIMITIVES

The additional function and terminal primitives have been added:

E_i

Double precision integer [-20,20]. Generation or mutation of E_i results in an integer, however, hybrid optimization can produce non-integer numbers.

E_d

Double precision number [-20,20].

Pow

Exponentiation.

REGISTERS

The registers allow for parameterization of the operators and optimization (e.g. mutation frequency, number of generations, etc.). The registers can be set by specifying the register and the value in the configuration file:

```
<Entry key="ec.pop.size">500/500/500/500</Entry>
<Entry key="ec.term.maxgen">100</Entry>
```

A list of registers and short descriptions is given below. If the value type of a register begins with U, the type is unsigned. If the value type is an array, individual elements are delimeted using a /.

ec.conf.dump <String> (def: "")

Filename used to dump the configuration. A configuration dump means that a configuration file is written with the evolver (including the composing operators) and the register (including the registered parameters and their default values). No evolution is conducted on a configuration dump. An empty string means no dump.

ec.elite.keepsize <UInt> (def: 1)

Number of individuals keep as is with strong n-elitism.

ec.hof.demesize <UInt> (def: 0)

Number of individuals kept in each deme's hall-of-**f**ame (best individuals so far). Note that a hall-of-**f**ame contains only copies of the best individuals so far and is not used by the evolution process.

ec.hof.vivasize <UInt> (def: 1)

Number of individuals kept in vivarium's hall-of-**f**ame (best individuals so far). Note that a hall-of-**f**ame contains only copies of the best individuals so far and is not used by the evolution process.

ec.init.seedsfile <String> (def: "")

Name of file to use for seeding the evolution with crafted individual. An empty string means no seeding.

ec.mig.interval <UInt> (def: 1)

Interval between each migration, in number of generations. An interval of 0 disables migration.

ec.mig.size <UInt> (def: 5)

Number of individuals migrating between each deme, at a each migration.

ec.pop.size <UIntArray> (def: 100)

Number of demes and size of each deme of the population. The format of an UIntArray is S1,S2,...,Sn, where Si is the ith value. The size of the UIntArray is the number of demes present in the vivarium, while each value of the vector is the size of the corresponding deme.

ec.repro.prob <Float> (def: 0.1)

Probability that an individual is reproducted as is, without modification. This parameter is useful only in selection and initialization operators that are composing a breeder tree.

ec.sel.tournsize <UInt> (def: 2)

Number of participants for tournament selection.

ec.term.maxfitness <Float> (def: 1)

Fitness value to reach before stopping evolution.

ec.term.maxgen <UInt> (def: 50)

Maximum number of generations for the evolution.

gp.cx.distrpb <Float> (def: 0.9)

Probability that a crossover point is a branch (node with sub-**t**rees). Value of 1.0 means that all crossover points are branches, and value of 0.0 means that all crossover points are leaves.

gp.cx.indpb <Float> (def: 0.9)

Individual crossover probability at each generation.

gp.init.maxargs <UIntArray> (def: 0/2)

Maximum number of arguments in GP tree. Tree arguments are is usually useful with ADFs (and similar stuff).

gp.init.maxdepth <UInt> (def: 5)

Maximum depth for newly initialized trees.

gp.init.maxtree <UInt> (def: 1)

Maximum number of GP tree in newly initialized individuals. More than one

FITNESS EVALUATION

The fitness evaluation in PM-Dreamer can be calculated using several different fitness statistics specified with the **-s** flag. The fitness in each case is given by *F*:

$$F = \frac{1}{1 + c \times s(\mathbf{e}, \mathbf{p})}$$

where **e** represents the energies and/or forces from the training set normalized by the number of distances used in the calculation of each energy/force and **p** represents those normalized values as calculated by a candidate individual. For the adaptive RMSD, c=1 and s(**e**,**p**) is the normalized root mean squared error between **e** and **p**. For the Pearson correlation coefficient, c=100 and s is given by the absolute value of the correlation coefficient between **e** and **p**. For OLS, ordinary least squares is performed to give the linear rescaling of **p** that results in the lowest RMSD with **e**. For 3-body potentials, the least squares problem is solved to optimize the linear combination of the 2 and 3 body functions that optimize the fit. In this case, s is this RMSD and **c** is 1. When using the Pearson correlation or OLS, the functions must be post-**p**rocessed to minimize the RMSD by solving analytically for new constants in the expression tree. This can be accomplished using **OLSCorrectOp** or **DreamerOp** as described below. The calculation of **p** according to the candidate expression is performed using one of several templates specified with the **-t** option. For all, the fitness calculation in the *beagle_config_file* file is specified using **EnergyOp**.

PAIR POTENTIALS (**-t** pair)

The pair potential, **pair**, is the default functional form used for fitness calculation. It is calculated as:

$$p = \frac{1}{n} \sum_{i=1}^{n} g\left(X_i\right)$$

where *X_i* is a single variable describing the particle pair (e.g. the inter-**p**article distance) and g is the function optimized using genetic programming. The fitness of the function is evaluated using a set of sample configurations for which the energies have been calculated. For example input file formats, see *pair* and *efxyz* below. In order to use this template, the X variable should be added to the primitive set:

```
<Primitive name="X"   bias="1"/>
```

PAIR POTENTIALS WITH FORCE (**-t** paird)

The pair potential with force, **paird**, is similar to **pair** with the exception that a particle force is supplied for a particle in each configuration allowing the potential function to be fit to both the energy and the force. When this style is used the fitness is one half the fitness statistic calculated for the energies plus one half the fitness statistic calculated for the forces.

When OLS is used as a fitness statistic, the least squares optimization is performed using only the energies - the force calculation is performed using the resulting formula. The Pearson correlation is calculated separately for the energies and forces - therefore it might not be possible to 'correct' the equations with a single set of optimized coefficients.

Here, the potential is calculated as described for the **pair** style, and the force is calculated as the negative gradient of the energy for a particle using forward finite-difference. This style requires an input format that supports forces; for an example, see efxyz below. The equations that result from the optimization will be in terms of the independent variable X which represents the interparticle distance for a pair as calculated from the supplied positions. Therefore, X should be added to the primitive set as described for **pair**.

PAIR POTENTIAL USING ONLY THE FORCE (**-t** pairf)

This template is similar to **paird** with the difference that only the forces are used in fitness evaluation. This style can therefore allow for much faster optimization followed by refinement by switching to style **paird**. Any energies specified in the input file are ignored.

PAIR POTENTIAL USING ONLY THE X-FORCE (**-t** pairf1)

This template is similar to **pairf** except that only the x component of the force is utilized for fitness evaluation.

TWO/THREE BODY POTENTIALS (**-t** twothree)

This template evaluates two summations for the potential energy and can be used to fit potentials that include a 2-body term and a 3-body term. The form for the expression is:

$$p = \frac{1}{n}\sum_{i}^{n} g\,(X_i) + \frac{1}{n}\sum_{i}^{m} h\,(R1_i, R2_{i,} A_i)$$

The sample data therefore consists of a set variables *X_1,...,X_n* that are evaluated in the first summation and a second set *R1_1,...,R1_m, R2_1,...,R2_m, and A_1,...,A_m* that are evaluated in the second summation, where *n* is not necessarily equal to *m*. For a 2/3-body potential, *X* might represent the interparticle distances in the 2-body part of the potential. For the 3-body part, *R1* and *R2* might represent the distances from particle 1 to particles 2 and 3 and *A* might represent the angle cosine between the corresponding vectors. Example input data file formats for this style include *twothree* and *efxyz* (below). In order to use this style, the variables X, R1, R2, and A must be added to the primitive set:

```
<Primitive name="X"   bias="1"/>
<Primitive name="R1"  bias="1"/>
<Primitive name="R2"  bias="1"/>
<Primitive name="A"   bias="1"/>
```

The equations for *g* and *h* are stored in the same expression tree where *g* is the left subtree of the root node and *h* is the right subtree of the root node. For this template, the root node is meaningless.

TWO/THREE BODY POTENTIALS WITH FORCE (**-t** twothreed)

This template evaluates the **twothree** potential style, but also evaluates the force for a single particle in each configuration in the fitness function. This is done in an identical manner to the **paird** potential style. Using the cutoff and particle positions, the vector **X** is calculated to contain all particle pairwise distances smaller than the cutoff. Likewise, for all particle triplets, the vectors **R1**, **R2** and **A** are calculated to contain the distances between the center atom and the other two atoms and the angle cosine between the corresponding vectors if the two distances are both smaller than the cutoff. As with the other twothree styles, the variables X, R1, R2, and A should be added to the primitive set.

As with paird, when OLS is used as a fitness statistic, the least squares optimization is performed using only the energies - the force calculation is performed using the resulting formula. The Pearson correlation is calculated separately for the energies and forces - therefore it might not be possible to ’correct’ the equations with a single set of optimized coefficients.

TWO/THREE BODY POTENTIALS USING ONLY FORCE (**-t** twothreef)

This template is similar to **twothreed** with the exception that only the force is used in the fitness evaluation. This allows for potentially faster optimizations and can be used to seed further runs that use the **twothreed** style. Any energies in the input data files are ignored.

DATA FILE TYPES

Several data file types are supported for specifying the energies and/or forces along with configuration data:

EFXYZ Data File Type

This data type supports configuration data in the form of atom positions with an option for periodic boundary conditions and a cutoff. Exactly 1 energy and 1 particle force are supplied per configuration. Zero(s) can be used in place of the energy or force if unknown and not used in the optimization (see fitness evaluation above). The file type supports only 1 particle type. The format for the input data file is:

```
# Comments for the input file

filetype efxyz
cutoff C
periodic p_x p_y p_z

 e  f_i  f_x  f_y  f_z  x_1  y_1  z_1  x_2  y_2  z_2 ...

...
```

First a cutoff is specified such that particle pairs with a distance greater than *C* contribute zero to the force and energy calculation. If *C* is negative, the cutoff is infinity. If the periodic keyword is present, periodic boundary conditions are used with box dimensions equal to *p_x*, *p_y* and *p_z*. Each of the following lines begins with an energy *e* followed by an index to a particle for which the force is computed, *f_i*. The first particle index is 1. This is followed by the Cartesian components of the force. Finally the Cartesian coordinates for each particle in the system are given.

EFIXYZ Data File Type

This file is similar to the efxyz format except that multiple particle types can be specified:

```
# Comments for the input file

filetype efixyz
cutoff C
periodic p_x p_y p_z

types N
 e  f_i  f_x  f_y  f_z  i_1  x_1  y_1  z_1  i_2  x_2  y_2  z_2 ...

...
```

where *N* is the number of particle types and *i* is the type for each particle. The valid range for particle types is [1..*N*]. See the multiple particle types section for more information on performing these types of optimizations.

PAIR Data File Type

The pair file type supports configuration input in terms of an energy and a single variable, X, that the potential is summed over to calculate an energy (e.g. X can be the interatomic distance used to compute the energy). The file type only supports 1 particle type for configurations and only 2-body potentials. Additionally, fitness evaluation is limited to templates that do not use particle forces. The format for the input file is:

```
# Comments for the input file

filetype pair

 e  X_1  X_2 ...
```

$e\ X_1\ X_2$ ...

...

Each line begins with an energy *e* and is followed by a variable number of data points for each pair in the configuration. Empty lines and lines beginning with # are ignored.

TWOTHREE Data File Type

This file type is used for potentials that perform a summation of some function over all pairs and a separate summation over all triplets. The template allows for a single variable, X, in the two-body equation (e.g. interatomic distance), and 3 variables (R1, R2, and A) for the 3-body equation (e.g. interatomic distances and triplet angle). The file type only supports 1 particle type and can only be used with 3-body fitness evaluation. Additionally, fitness evaluation is limited to templates that do not use particle force. The format for the input data file is:

Comments for the input file

filetype twothree

e **TWO** X_1 ... X_n **THREE** $R1_1\ R2_1\ A_1$ ... $R1_m\ R2_m\ A_m$

...

where *e* is the energy of the configuration.

PARALLEL PM-DREAMER

PM-Dreamer can be run in parallel using an island model. In serial, PM-Dreamer uses the Open Beagle model allowing for multiple populations with individual movement according to migration operators. In parallel PM-Dreamer allows for multiple islands, 1 per process, to be run. Each island can contain multiple populations with migration controlled by the standard operators. Migration between the islands is controlled by additional operators which are described below. The output for each island is written separately to the files gp_force_0.log, gp_force_1.log, ... and gp_force_0.obm(.gz), gp_force_1.obm(.gz), ... It should be noted that in the current implementation, random seeds only produce the same output when run on the same number of processors. When running in parallel, the MPITerminateOp should be used to to assure proper termination of all processes in a run. The additional operators available for parallel execution are:

MigrationMPIOp

Each time *ec.mig.mpi_interval* generations passes, *ec.mig.mpi_size* individuals from each population on an island migrate to a randomly chosen island and are replaced with immigrants from a second randomly chosen island. The random islands are chosen such that all islands will participate in migration at each iteration. The operator does not perform migration between populations on the same island. This can be achieved using standard migration operators in addition to MigrationMPIOp

HFCompMPIOp

This implements a distributed parallel algorithm for the Hierarchical Fair Competition inspired by the work of Hu and Goodman. (Similar to the serial HierarchicalFairCompetitionOp). This operator should not be used with the serial HierarchicalFairCompetitionOp operator. In this algorithm, a fitness threshold is chosen such that any individuals from a population with index *i* will migrate to the population *i*+1 if their fitness is better than the fitness threshold for that population *i*+1. If any

populations has excess individuals following migration, the least fit individuals are killed off. Random individuals are added to account for any shortages. This promotes a hierarchy of populations where the fitness of the best individuals improves with the population index. The migration occurs through all populations on a single island followed by migration of individuals of the last population of one island to the first population of another. In order to achieve parallel efficiency, their is a 1-**s**tep lag from the time individuals migrate out of an island to the time they arrive at the next. The fitness thresholds for the populations can be set in 2 ways. In the default, ec.hfc.first is set to -1 and the fitness threshold for a population is set to a value where the threshold is greater than ec.hfc.percentile of the population. For example, if ec.hfc.percentile is 0.85 the fitness threshold for a population is set to the value of the individual whose fitness is worse than only 15% of the population. In the case, the fitness thresholds are adaptive. In the second approach, the fitness thresholds are fixed. ec.hfc.first (float greater than 0 and less than 1) is set to the fitness threshold of the first population accepting incoming individuals. The thresholds for the subsequent populations are increased according to ec.hfc.scale (described below)to allow for thresholds up to but less than 1.0

GP-StatsCalcFitSimpleMPIOp

This operator can replace GP-StatsCalcFitnessSimpleOp to replace Vivarium statistics for a single island with Vivarium statistics for all processes in the log file on process 0. The best hall-of-**f**ame individual from all processes is also stored in the milestone file for process 0. When using this operator, analysis of the log file generated on process 0 should be all that is necessary under most circumstances. This operator will also output fitness information and a "pretty" representation of the best individual at each generation to stdout.

MPITerminateOp

This signals the application to terminate execution of all processes whenever a single island is terminated by any of the termination operators. This operator also delays termination until all demes are evaluated for a given generation. This allows proper communication of statistics and end-of-**r**un operations such as simplification and OLS correction to occur In order to work properly, the operator should be placed following any other termination operators.

TermMaxTimeOp

Terminate after ec.term.maxtime minutes have passed. If compiled with MPI, this is the MPI wall time. Otherwise, this is the time calculated using c_time clock(). If set to zero, the operator is ignored. This operator will also log the time at each generation to stdout

The registers available for parallel execution are:

ec.mig.mpi_interval

The number of generations that must pass before a migration between islands occurs.

ec.mig.mpi_size

The number of individuals that migrate from each population of each island.

ec.term.maxtime

Terminate the evolution after this many minutes (default 60).

ec.hfc.percentile

Percentile of fitness measure to use as HFC migration threshold of next population. For example, a threshold of 0.85 means that the fitness used as threshold to accept migrant into following population is taken as the fitness of the individual that is better than 85% of the other individuals in its population. Default is 0.85. This value is ignored if ec.hfc.first is positive

ec.hfc.first

If negative, adaptive thresholds are used for HFC according to ec.hfc.percentile. If positive, the register must be greater than 0 and less than 1.0. The thresholds for the populations in HFC are then set evenly spaced fixed values between *first* and 1.0.

ec.hfc.scale

This parameter is used to adjust how the fitness thresholds of populations are scaled if adaptive thresholds are not used (ec.hfc.first>0). The ratio between the fitness thresholds of populations is given by ec.hfc.scale to create a geometric series between ec.hfc.first and 1.0. If ec.hfc.scale is 1, the fitness thresholds are evenly spaced. If ec.hfc.scales is >1, more of the fitness thresholds are at lower fitnesses. If <1, more are at higher fitnesses. The default value is 1.

ec.hfc.interval

Interval between each hierarchical fair competition migration, in number of generations. An interval of 0 disables HFC migrations. Default is 1.

ec.hfc.copy_num

This flag controls the number of islands that have the same fitness thresholds. For example, a 4 processor job can have fitness thresholds of {0, 0.25, 0.5, 0.75} when copy_num is 1 or fitness thresholds of {0, 0, 0.5, 0.5} when copy_num is 2. For the latter case, an individual on island 0 or 1 who exceeds the fitness threshold of 0.5 will migrate to either population 3 or 4 randomly. When copy number is greater than 1, migration between islands with the same fitness threshold can be allowed with the use of MigrationMPIOp and a ec.mig.mpi_split register value equal to copy_num. This register can also be set from the command line. Default is 1.

ec.mig.mpi_split

When this register is greater than 1, the islands are split into groups of size ec.mig.mpi_split. Each group contains a sequential ordering of islands according to process rank. Migration according to the MigrationMPIOp will then only occur within a group. This can be used with HFCCompMPIOp to restrict migration only between populations with the same fitness threshold. Default is 0

HYBRID PM-DREAMER

PM-Dreamer supports hybrid optimization of functional forms, allowing for local optimization of constants in the expression tree. This is accomplished by adding the **GP-HybridOptOp** operator. The registers that parameterize the operator include:

gp.hybopt.indpb

The frequency with which hybrid optimization is performed on an indivual. The default value is 0.05.

gp.hybopt.primit

The name for the constants in the tree that are optimized. Default is E.

gp.hybopt.type

The type of optimization to be performed. If the value is 0, all constants in an expression tree are optimized using multidimensional Nelder/Mead Simplex algorithm. If the value is nonzero, a random constant in the expression tree is optimized using 1D minimization (also with Simplex). The default value is 1.

gp.hybopt.maxi

The maximum number of iterations of local optimization to be performed. Default is 10.

gp.hybopt.simplify

If nonzero, function simplification is performed before each optimization. See SimplifyOp. Default is 0.

gp.hybopt.mtypes

If nonzero, optimization is also performed on the E_p and E_t constants used for multiple particle types. See MULTIPLE PARTICLE TYPES section below for more details. Default is 1.

Please see Function Simplification for additional registers that control simplification during hybrid optimization.

An analytic optimization is also available that utilizes least squares for function optimization. For a 2-body potential function *f* the function is modified to give *f'=af+b* where *a* and *b* are optimized to result in the lowest RMSD. For a 3-body potential *f+g*, four constants are optimized to give *af+b+cg+d*. This is accomplished with the **GP-OLSCorrectOp** operator. Because this operator increases the function size at each operation, it is recommended that it only be performed once (e.g. at the end of a run). **DreamerOp** described below is one way to accomplish this. The registers for **GP-OLSCorrectOp** are:

gp.olscorrect.name

The name of the primitives added that store constant values. Default is E.

gp.olscorrect.indpb

The probability of operation on an individual. Default is 1.0

gp.olscorrect.interval

The interval in generations between operations. Default is 20

FUNCTION SIMPLIFICATION

PM-Dreamer supports function simplification to reduce tree size. This is accomplished by evaluating each non-**t**erminal node of the tree for the set of input values. If the range of answers for the set of evaluations is smaller than a specified threshold, the subtree is replaced with a constant value. Likewise, if a subtree evaluates to be identical to an input variable, it is replaced with that variable. This is accomplished by adding the **GP-SimplifyOp** operator. The registers that parameterize the operator include:

gp.simplify.indpb

The frequency with which simplification is performed on an indivual. The default value is 1.0.

gp.simplify.name

The name of the ephemeral for constants used to replace subtrees. Default is E.

gp.simplify.eps

The threshold for determining invariance. The default is 1e-20.

gp.simplify.maxe

The maximum number of evaluations used to determine invariance. The evaluations are performed using the same input data that is used for fitness evaluation. If 0, all input data are used. Otherwise, the first maxe function evaluations are performed. The default is 0.

gp.simplify.interval

Interval in generations between each simplification operation. An interval of 0 disables simplification. Default is 20.

You can also perform simplification on an individual before hybrid optimization. In this case, the <GP-SimplifyOp> operator does not need to be specified explicitly in the configuration file. However the registers that control simplification should still be specified (aside from gp.simplify.indpb which is not used).

DREAMER OPERATOR

The Dreamer operator **DreamerOp** is utilized to perform post-**p**rocessing of the best individual from a run. Additionally, the operator includes the functions of **MPITerminateOp** and **GP-StatsCalcFitSimpleMPIOp**. At the end of a run the best individual is simplified (**GP-SimplifyOp**) and optimized (**GP-HybridOptOp**) using the user-**s**pecified fitness statistic. OLS optimization is then performed (**GP-OLSCorrectOp**). This is followed by a second round of simplification and optimization using AdaptiveRMSD as the fitness statistic. For parallel runs, the best individual from all processes is moved to the hall-of-**f**ame on process 0. The user-**s**pecified fitness, functional form, and RMSD for the energies and/or forces is reported. As with **GP-StatsCalcFitSimpleMPIOp**, this function will add monitoring of the fitness and functional form of the best individual during the run. Because the operator incorporates **MPITerminateOp**, it should be placed following any other termination operators. The following registers (described above) can be specified when using **DreamerOp**: *gp.simplify.name*, *gp.simplify.eps*, *gp.simplify.maxe*, *gp.olscorrect.name*, *gp.hybopt.primit*. **gp.hybopt.type** is set to 0 and gp.hybopt.maxi is set to 100 when performing optimization *within* **DreamerOp**. **GP-StatsCalcFitSimpleMPIOp** and **MPITerminateOp** do not need to be specified in the input parameter file when using **DreamerOp**. **GP-HybridOptOp** and **GP-SimplifyOp** should be specified if you wish to perform these operations during the run.

RESTARTING OPTIMIZATIONS

Simulations can be restart using the **-c** flag. This requires that the ReadRestartOp operator be present in the *beagle_config_file*. The ReadRestartOp replaces the MilestoneReadOp in OpenBeagle. When restarting, the individuals and the generation number are read. The data, template_style, and parameters from the previous run are not read in. This allows the user to continue a run with new data, template_style, and/or configuration parameters. When restarting a run with new data or a new template_style, the fitness of all individuals

are recalculated and the Hall of Fame individuals are updated with any changes that result from the new fitness evaluation. Because the restart will start using the last generation from the milestone files, the ec.term.maxgen register may need to be increased to allow for a larger number of generations. When restarting in parallel, if a smaller number of processors is used, the individuals from the higher rank processes will be thrown out.

VECTORIZATION

PM-Dreamer allows for vectorized evaluation of expression trees (which is now the default). Vectorization can be disabled by using the **-z** flag. When vectorization is enabled, the expression tree for a given individual needs to be parsed only a single time using the vector(s) of values necessary for energy/force calculation. This provides an improvement in speed because it prevents multiple parsing of the same expression tree and the potential for SIMD compiler optimizations. The configuration file does not need to be changed to utilize vectorization; internal replacements of the standard primitives and fitness operators are performed to allow vector math operations to be performed. Although the runs with vectorization should produce identical results, changes due to finite precision and the order of summation operations can result in different results. Because certain Open Beagle primitives have the argument types hard**-c**oded, vectorization is left as an option to aid in compatability with future versions.

MUTIPLE PARTICLE TYPES

PM-Dreamer allows for optimization using multiple particle types as specified with the **efixyz** file format. Output is given to standard out at the beginning of a simulation with the number of interactions for each combination of particle types (for example):

```
Pair_Particle_Type Count
-----------------------
1-1 77
2-1 154
2-2 69

3body_Particle_Type Count (Center First)
----------------------------------------
1-1-1 33
1-2-1 88
1-2-2 33
2-1-1 44
2-2-1 66
2-2-2 36
```

Optimization using multiple particle types requires the addition of special primitives in order to be effective. For 2-body potentials, E_p can be added to optimize constants for each particle type interaction:

```
<Primitive name="E_p" bias="1"/>
```

The primitive E_p will be described in any equation as a vector of constants in the same order as the type counts written at the beginning of the optimization. For 3-body combinations, the primitive E_t can also be added:

```
<Primitive name="E_t" bias="1"/>
```

In equations written to standard out, each occurance of an E_p or E_t primitive is replaced with a lower case variable (starting with the letter a). The vector corresponding to each variable is written on the following lines (for example):

2-Body: (a*X)/1.7441+b
a=[-4.0801 -14.4792 -16.8834]
b=[-8.4213 -21.0239 253.8123]
3-Body: exp(R1)*(2.3695/c)
c=[-3.2828 11.8393 17.4884 -15.1295 9.2678 19.6123]

Therefore it is recommended to use only upper case names for any other primitives when using multiple particle types. The 2- and 3- body contant vectors have their own mutation operators and registers. The operators for 2- and 3- body (respectively):

<GP-MutationEphemeral2COp/>
<GP-MutationEphemeral3COp>

and the registers are:

<Entry key="gp.mute2c.indpb">0.05</Entry>
<Entry key="gp.mute2c.primit">E_p</Entry>
<Entry key="gp.mute3c.indpb">0.05</Entry>
<Entry key="gp.mute3c.primit">E_t</Entry>

If hybrid local search is used during the optimization and gp.hybopt.mtypes is non-**z**ero (default), local optimization of contant vectors will automatically be performed in addition to the primitive specified by the gp.hybopt.primit register. If gp.hybopt.type register is 0 all constants in the vectors will be optimized. If it is 1 all of the contants in 1 chosen vector might be optimized. Optimization of multiple particle types is currently limited to vectorized simulations.The register gp.simplify.maxe is ignored when multiple particle types are used; in this case, the entire dataset is used to determine invariance.

LIMITATIONS

Fitness metrics in PM-Dreamer are limited to those that generate floating points >=0 and <=1 with 1 representing a perfect match to machine precision. For certain functions, the input variables are only checked during initialization and therefore cannot change in the middle of a run. The following epsilon values are currently hard coded in PM-Dreamer:

1e-8

Epsilon for forward finite difference.

1e-8 and 1e16

Are the minimum and maximum ranges for test data evaluated using a correlation coefficient or OLS.

The primitive names E_p and E_t should only be used in optimizations involving multiple particle types as described above. The register gp.simplify.maxe is ignored when multiple particle types are used; in this case, the entire dataset is used to determine invariance.

AUTHORS

W. Michael Brown